An Approximation-Based Data Structure for Similarity Search*

Roger Weber
Institute of Information Systems
ETH Zentrum, 8092 Zurich, Switzerland
weber@inf.ethz.ch

Stephen Blott†
Bell Laboratories (Lucent Technologies)
700 Mountain Ave, Murray Hill, NJ 07974, USA
blott@research.bell-labs.com

Abstract

Many similarity measures for multimedia retrieval, decision support, and data mining are based on underlying vector spaces of high dimensionality. Data-partitioning index methods for such spaces (e.g. grid-files, quad-trees, R-trees, X-trees, etc.) generally work well for low-dimensional spaces, but perform poorly as dimensionality increases—a phenomenon which has become known as the 'dimensional curse'.

In this paper, we first provide an analysis of the nearest-neighbor search problem in high-dimensional vector spaces. Under the assumptions of uniformity and independence, we establish bounds on the average performance of three important classes of data-partitioning techniques. We then introduce the *vector-approximation file* (VA-File), a method which overcomes the difficulties of high dimensionality by following not the data-partitioning approach of conventional index methods, but rather a filter-based approach. A VA-File contains a compact, geometric approximation for each vector. By first scanning these smaller approximations, only a small fraction of the vectors themselves must be visited. Thus, the VA-File acts as a simple filter, much as a signature file is a filter. Performance is evaluated on the basis of both synthetic and real data sets, and compared to that of the R*-tree and the X-tree. We show that performance does not degrade, and even improves with increased dimensionality. Both our analytical and our experimental results suggest that the VA-File is generally the preferred method for similarity search over moderate and large data sets with dimensionality in excess of around ten.

*This work has been partially funded in the framework of the European ESPRIT project HERMES (project no. 9141) by the Swiss *Bundesamt für Bildung und Wissenshaft* (BBW, grant no. 93.0135).

†The work on which this paper is based was carried out in part while Stephen Blott was a member of the Database Research Group at ETH, Zurich.

1 Introduction

One of the important distinctions between traditional database systems and systems for multimedia data, decision support, and data mining is the need not just for boolean search, but also for similarity search. By similarity search we mean the problem, given a data set, of finding the k objects 'most similar' to a given object. This has led to the development of many methods for measuring 'similarity' between, for example, documents, images, faces, finger prints, X-rays, time series, etc. Typically, however, similarity is measured not on objects directly, but rather on abstractions of objects termed *features*. Though the feature-extraction process itself is application specific, the resulting features can frequently be viewed as points in high-dimensional vector spaces [SO95, FSN+95, Csi95, Csi97]. We refer to such features as *feature vectors*. The dimensionality of features vectors can be moderate, such as 4–8 in [FSN+95] or 45 in [SO95], but it can also become quite large (315 in a recently proposed color index [Dim97], or over 900 in some astronomical indexes [Csi95, Csi97]). The similarity of two feature vectors is measured as the distance between them. As such, similarity search corresponds to a search for the nearest neighbor (or neighbors) within the space of the feature vectors.

The conventional approach to nearest-neighbor search is to utilize some form of multidimensional access method (such as a grid-file [NHS84], a quad-tree, a k-d-tree, an R-tree [Gut84], R*-tree [BKSS90], TV-tree [LJF94], X-tree [BKK96], SR-tree [KS97] or M-tree [CPZ97], etc.). In general terms, these methods work by partitioning the data space, clustering data (or references to data) according to the partitioning in which they lie, and using the partitioning to prune the search space for queries. Therefore, we refer to these methods as *data-partitioning methods*. Several surveys provide background and analysis of these methods [BF79, Sam89, Fal96]. Unfortunately, while these methods generally perform well at low dimensionality, performance degrades as dimensionality increases—a phenomenon which has been termed the 'dimensional curse'. Practical evaluations have shown that whenever the (inherent) dimensionality exceeds around 10, then a simple sequential scan through the vector data itself outperforms the data-partitioning methods mentioned above. This result has been reported for R*-trees and X-trees [BKK96, BBB+97], and also for the SR-tree [KS97].

In this paper, we address this issue in two ways. The first part of the paper provides a formal analysis of the problem. Under the assumptions of uniformity and independence, we establish bounds on the average performance of data-partitioning methods. We show that these approaches must ultimately fail if dimensionality becomes larger than around 10 (where by 'fail' we mean that a simple sequential scan of the vector data is expected to offer better performance, on average). This result is achieved for three important classes of data-partitioning methods: those whose minimum bounding regions (MBRs) are hyper-cubes, those whose MBRs are hyper-spheres, and (as the limiting case) those whose MBRs are lines. Although of questionable value as a practical method, the latter case is of theoretical

importance since it provides a bound on the average performance of any possible data-partitioning method.

In the second part of the paper we propose a new, flat access method based on a *vector-approximation file* (VA-File). Rather than partitioning the data space as most existing methods do, the VA-File is a flat file containing a compact, geometrical approximation for each vector. Typically, the approximations occupy only between 10% and 20% of the space of the vectors themselves. Based only on these smaller approximations, the vast majority of the vectors can be excluded from a search. Thus, the VA-File is used as a simple filter, much as a signature file is a filter [Fal85, FC87]. Based on the VA-File, we describe two nearest-neighbor search algorithm, one of which has an optimality property for all but a few highly-improbable cases.

The main advantage of the new method is that the VA-File retains good performance as dimensionality increases. Indeed, we show that performance can even improve with increased dimensionality, such that VA-File outperforms data-partitioning methods by an order of magnitude or more. However, further important advantages also follow from the VA-File's simple flat structure. Distribution, parallelization, concurrency control and recovery (which can be complicated for tree-based structures) are simplified for the flat, array-like structure of the VA-File. Moreover, the VA-File also supports weighted search, thereby allowing relevance feedback to be incorporated. Relevance feedback can have a significant impact on search effectiveness (in terms of precision and recall) for many applications.

The remainder of this paper is structured as follows. Section 2 investigates the dimensional curse problem from an analytical perspective. Section 3 describes the VA-File, and Section 4 provides a performance evaluation. Section 6 discusses the method and concludes.

2 Nearest-Neighbor Search in High Dimensional Vector Spaces

This section establishes a number of analytical results as to the average performance of nearest-neighbor search in partitioned organizations of vector spaces. Broadly speaking, our analysis proceeds as follows. Under the assumptions of uniformity and independence, we compute the expected nearest-neighbor distance for a query. Given this distance, a nearest-neighbor query can be reformulated as a spherical range query, such that both the nearest-neighbor query and the reformulated range query visit exactly the same set of blocks. Thus, the average 'cost' of a search can be measured by counting the blocks which intersect the sphere. We then investigate three important classes of data-partitioning methods: those whose *minimum bounding regions* (MBRs) are hyper-cubes, those whose MBRs are hyper-spheres, and (as the limiting case) those whose MBRs are lines. In each of these cases, we establish closed formulae for the average cost of nearest-neighbor (NN) search.

Our analysis is based on the following assumptions:

1. We assume that data is uniformly distributed within the data space, and that dimensions are independent from one-another (these key assumptions are discussed below).

2. We assume that data is partitioned into 'blocks', and make only weak assumptions about the partitioning algorithm or strategy. As such our analysis is applicable to any of the existing index methods mentioned above, and also to clustering methods based on space-fitting curves (Z-ordering, Hilbert curve, etc. [Sam89]).

3. We assume that a data-partitioning method works 'well' if on average less than 20% of blocks must be visited, and 'fails' if more than 20% of blocks must be visited. This assumption is based on the fact that a direct sequential scan of the data can expect a large performance boost simply from the sequential nature of its IO requests. Although a factor of 10 for this phenomenon is frequently assumed elsewhere (and was observed in our own PC and workstation environments), we assume a more conservative factor of only 5 here.

The uniformity and independence assumptions (Assumption 1, above) are generally questionable in practice. Real data sets are frequently not uniformly distributed, and may exhibit correlation between dimensions. However, it has been shown that 'fractal dimensionality' can be a useful measure for predicting the performance of data-partitioning access methods [FK94]. In some sense, the fractal dimensionality of a data set is a measure of a data sets inherent dimensionality. A data set consisting of a line in d-dimensional space is still a line, so its fractal dimensionality is 1. Similarly, a data set consisting of a plane in d-dimensional space is still a plane, so its fractal dimensionality is 2. On the other hand, the fractal dimensionality of a (sufficiently-large) d-dimensional data set which conforms to the uniformity and independence assumptions above is d. Therefore we conjecture that, for d-dimensional data sets, the results obtained here under the uniformity and independence assumptions generally apply also to arbitrary higher-dimensional data sets of fractal dimension d. This conjecture appears reasonable, and is supported by the experimental results of Faloutsos and Kamel for the effect of fractal dimensionality on the performance of R-trees [FK94].

2.1 Basic Definitions

For simplicity, we focus here on the widely-used L_2 metric (Euclidean distance), although the structure of our analysis might equally be repeated for other metrics (such as L_1 or L_∞). Where appropriate, our notation and parts of our analysis follow that of Berchtold et al [BBKK97].

The nearest neighbor to a query point Q in a d-dimensional space may be defined as follows:

d	number of dimensions	N	number of data points
k	number of NNs to return	$\Omega = [0,1]^d$	data space
Q	query point	$nn(Q)$	NN to query point Q
$nn^{dist}(Q)$	NN-distance of query point Q	$sp^d(C,r)$	d-dim hyper-sphere
$E(nn^{dist})$	expected NN-distance	M^{visit}	number of visited blocks
		m	number of points / block

Table 1: Notational summary

Definition 2.1 (Nearest Neighbor (NN), NN-distance, NN-sphere)
Let $\mathcal{D}$ be a set of d-dimensional points. Then the nearest neighbor to the query point Q is the data point $nn(Q) \in \mathcal{D}$, which lies closer to Q than any other point in $\mathcal{D}$:

$$nn(Q) = \left\{P \in \mathcal{D} \mid \forall P' \in \mathcal{D}: \ \|P - Q\|_2 \ \leq \ \|P' - Q\|_2\right\}$$

where $\| \bullet - \bullet \|_2$ denotes Euclidean distance. The nearest neighbor distance $nn^{dist}(Q)$ is the distance between Q and its nearest neighbor $nn(Q)$:

$$nn^{dist}(Q) \ = \ \|nn(Q) - Q\|_2$$

and the NN-sphere $nn^{sp}(Q)$ is the sphere with center Q and radius $nn^{dist}(Q)$. □

Analogously, one can define the k-nearest neighbors to a given query point Q. Then $nn^{dist,k}(Q)$ is the distance of the k-th nearest neighbor, and $nn^{sp,k}(Q)$ is the corresponding NN-sphere.

We assume that data points are assigned to blocks such that each data point belongs to a single block. Moreover, we assume some minimum bounding region (MBR) is defined for each block such that all points in the block are contained within the MBR. We do not require that the MBRs are disjoint, although they may be.

Definition 2.2 (MBR) *The minimum bounding region of a block is the minimum region of a specified geometrical type (e.g. hyper-cube, hyper-sphere, line) that contains all the data points of the block.* □

We assume that the data space Ω is the unit hyper-cube: $\Omega = [0,1]^d$. Let $Q \in \Omega$ be a query point, and let $sp^d(Q,r)$ be the hyper-sphere around Q with radius r. Then, under the uniformity and independence assumptions, for any point $P \in \mathcal{D}$, the probability that $sp^d(Q,r)$ contains P is equal to the volume of that part of $sp^d(Q,r)$ which lies inside the data space. This volume can be obtained by integrating a piecewise defined function over the entire data space:

$$Vol\left(sp^d(Q,r) \cap \Omega\right) = \int\limits_{P \in \Omega} \left(\begin{cases} 1 & \text{if } \|C - P\|_2 \leq r \\ 0 & \text{otherwise} \end{cases} \right) dP$$

As dimensionality increases, this integral becomes difficult to evaluate. Fortunately, good approximations for such integrals can be obtained by the Monte-Carlo method.

2.2 Expected Nearest Neighbor Distance

The expected NN-distance between a query point Q and its nearest neighbor $nn(Q)$ is a critical factor for all data-partitioning methods. If this distance is known, then a NN-query can be transformed into a range query, and search costs can be estimated.

Let $P(Q,r)$ be the probability, that the NN-distance $nn^{dist}(Q)$ is at most r (i.e. the probability that $nn(Q)$ is contained in $sp^d(Q,r)$). This probability distribution function can be expressed in terms of its complement, that is, in terms of the probability that all N data points lie outside the hyper-sphere:

$$P(Q,r) = 1 - \left(1 - Vol\left(sp^d(Q,r) \cap \Omega\right)\right)^N$$

and the corresponding probability density function $p(Q,r)$ is the derivative:

$$p(Q,r) = \frac{dP(Q,r)}{dr}$$

The expected nearest-neighbor distance for a query point Q can be obtained by integrating over all radii r:

$$E(Q, nn^{dist}) = \int_0^\infty r \cdot p(Q,r) dr$$

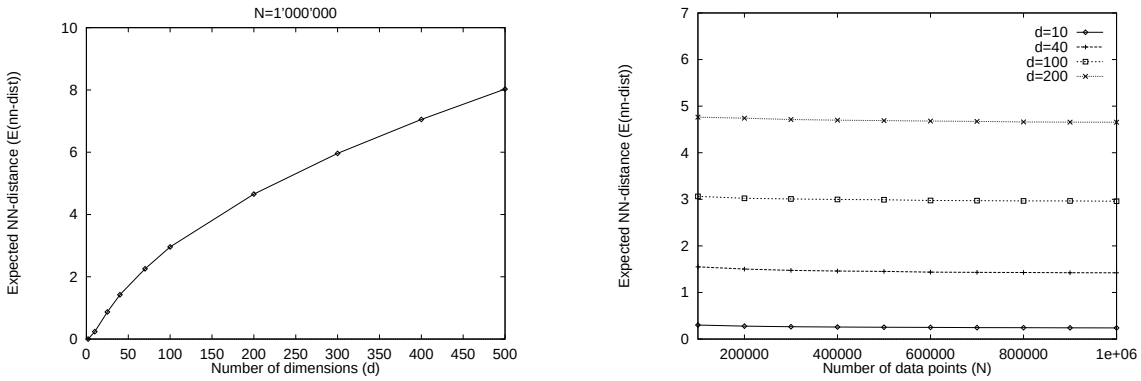
Finally, the expected NN-distance $E(nn^{dist})$ for any query point in the data space is the average of $E(Q, nn^{dist})$ over all possible points Q in Ω (since Ω is the unit hyper-cube, the dominator of this average is 1):

$$E(nn^{dist}) = \iint_{Q \in \Omega} E(Q, nn^{dist}) dQ$$

Based on this closed formula, we used the Monte-Carlo method to estimate nearest-neighbor distances. Figure 1 shows this distance as a function (a) of dimensionality, and (b) of the number of data points. Notice, that $E(nn^{dist})$ can even become larger than the length of the data space itself.

The main conclusions we draw at this stage are:

1. The NN-distance grows steadily with dimensionality (this result can also be derived from the Central Limit Theorem), and
2. Beyond trivially-small data sets, NN-distances decrease only slightly as the size of the data set increases.



(a) (b)

Figure 1: Expected nearest neighbor distance as a function: (a) of the dimensionality; and (b) of the number of data points

The average value for $E(nn^{dist})$ grows because Ω becomes sparser and sparser as dimensionality increases. In order to maintain a constant density in the data space, the number of data points would have to grow exponentially with the number of dimensions. This is clearly not the case for real data sets. As a consequence, objects are widely scattered and, as we shall see, the probability of being able to identify a good partitioning of the data space diminishes.

2.3 Search Costs: Expected Number of Intersected Blocks

We now establish bounds on the expected number of blocks which must be accessed for a nearest-neighbor search. Our results depend upon the geometry of blocks' MBRs. In particular, we derive closed formulae for the expected number of blocks accessed in the cases that MBRs are either hyper-cubes (e.g. R*-tree, X-tree), are hyper-spheres (e.g. M-tree, VP-tree) or, in the limiting case, are lines (with two points per block). For disk-resident databases, the number of blocks which must be accessed is a measure of the amount of IO which must be performed, and hence of the 'cost' of a query. We first introduce the general method, then study the three special cases in the following sections.

2.3.1 The General Method

A nearest-neighbor search algorithm is optimal if the blocks visited during the search are exactly those whose MBRs intersect the NN-sphere. Such an algorithm has been proposed by Hjaltson and Samet [HS95], and shown to be optimal by Berchtold et al. [BBKK97]. This algorithm visits blocks in increasing order of their minimal distance to the query point, and stops as soon as a point is encountered which lies closer to the query point than all remaining blocks. In the general case, the

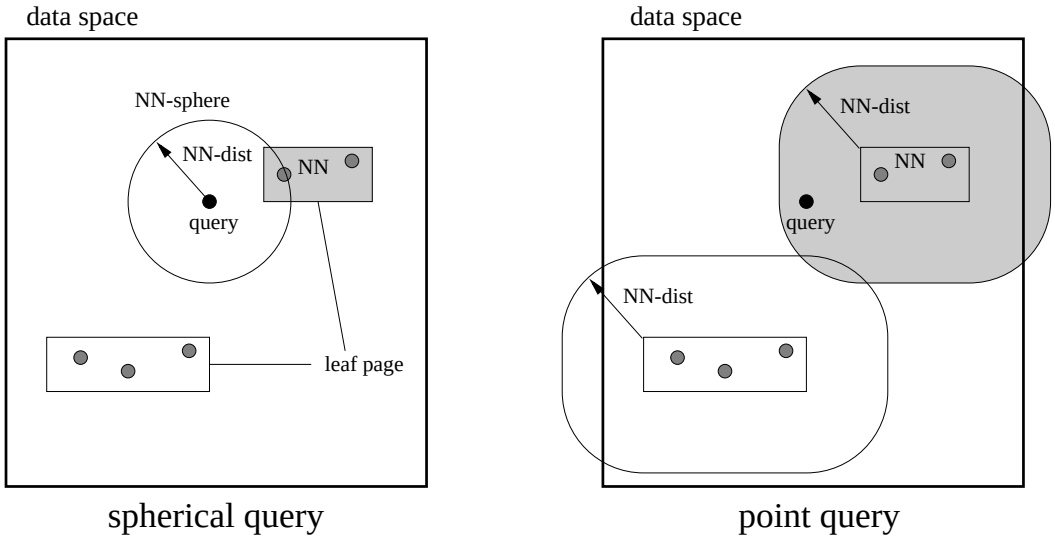


Figure 2: The transformation of a spherical query into a point query

algorithm stops after having found k points that lie closer to the query than the remaining blocks.

Given this optimal algorithm, let M^{visit} denote the number of blocks visited. Then M^{visit} is equal to the number of blocks which intersect the nearest neighbor sphere $nn^{sp}(Q)$ with, on average, the radius $E(nn^{dist})$. To estimate M^{visit}, we transform the spherical query into a point query by the technique of Minkowski sum (following [BBKK97]—see Figure 2). A block i is extended by moving the center of the NN-sphere over the surface of its minimum bounding region (mbr_i). The result is an enlarged object $MinkSum\left(mbr_i, E(nn^{dist})\right)$. The volume of the part of this region which is within the data space corresponds to the fraction of all possible queries in Ω whose nearest neighbor sphere intersects that block. Therefore, the probability that the i-th block must be visited is given by:

$$P_{visit}(i) = Vol\left(MinkSum\left(mbr_i, E(nn^{dist})\right) \cap \Omega\right)$$

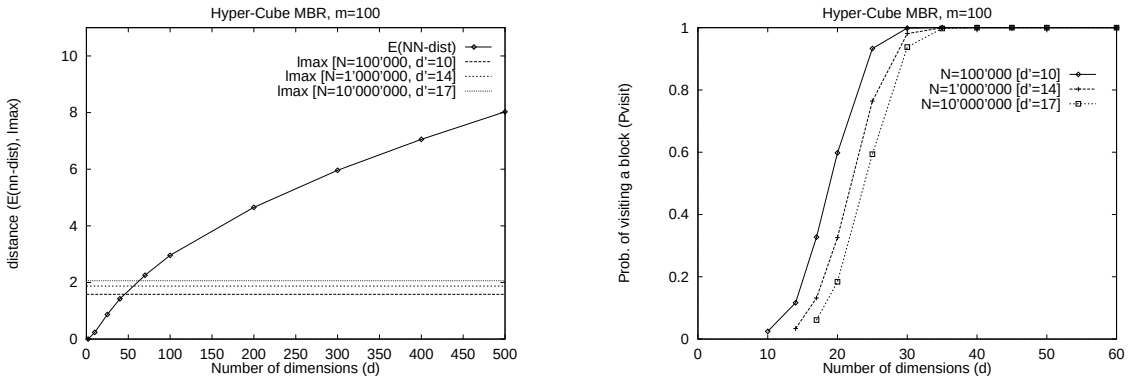
Finally, the expected number of blocks which must be visited is given by the sum of this probability over all blocks. If we assume m objects per block, then we arrive at:

$$M^{visit} \;=\; \frac{N}{m} \cdot P_{visit}(i) \;=\; \frac{N}{m} \cdot Vol\left(MinkSum\left(mbr_i, E(nn^{dist})\right) \cap \Omega\right)$$

This open formula depends upon the geometry of mbr_i. Next, we establish closed lower bounds for M^{visit} under the assumptions that MBRs are either hyper-cubes, are hyper-spheres, or are lines.

2.3.2 Hyper-Cube MBRs

Index methods such as R*-tree, X-tree and SR-tree (see also next section) use hyper-cubes to bound the region of a block. Usually, splitting a node results in two new, equally-full partitions of the data



(a) (b)

Figure 3: Hyper-cubes: (a) the maximum distance l_{max} to a block, and (b) the probability of visiting a block.

space. Assume, that each dimension is split at least once. Since after each split the number of points in the partitions is halved, there must be more than 2^d data points in the index. However, even at moderate dimensionality this number far exceeds the size of real databases. Thus, assuming an average of m data points are assigned to each block, the number of split dimensions (d') is smaller than the total number of dimensions, and we obtain an upper bound:

$$d' \leq \left\lceil \log_2 \frac{N}{m} \right\rceil$$

Furthermore, each dimension is split at most once, and, since data is distributed uniformly, the split position is always at $\frac{1}{2}$. Hence, the MBR of a block has d' sides with a length of $\frac{1}{2}$, and $d - d'$ sides with a length of 1. For any block, let l_{max} denote the maximum distance between an arbitrary point in the data space Ω and the MBR of the block. Then l_{max} is given by the equation:

$$l_{max} \;=\; \frac{1}{2}\sqrt{d'} \;=\; \frac{1}{2}\sqrt{\left\lceil \log_2 \frac{N}{m} \right\rceil}$$

Assume that each block contains 100 data points (i.e. $m = 100$). If we overlay l_{max} on the expected NN-distance (see Figure 3 (a)), we observe that l_{max} becomes smaller than $E(nn^{dist})$ if dimensionality is larger than around 50. In other words, if we enlarge the MBR by the expected NN-distance according to the Minkowski sum, the resulting region covers the entire data space. The probability of visiting a block is 1. Consequently, all blocks must be accessed, and even nearest-neighbor search by an optimal search algorithm degrades to a (poor) scan of the entire data set. Figure 3 (b) shows the probability of accessing a block during a NN-search for different database sizes, and different values of d'. The graphs are only plotted for dimensions above the number of split axes, that is 10, 14 and 17 for 10^5, 10^6 and 10^7 data points, respectively. Depending upon the database size, the 20% threshold is

exceeded for dimensionality greater than around $d = 15$, $d = 18$, and $d = 20$. Based on our earlier assumption about the performance of scan algorithms, these values provide upper bounds on the dimensionality at which any data-partitioning method with hyper-cubic MBRs can be expected to perform 'well', on average. On the other hand, for low-dimensional spaces (that is, for $d < 10$), there is considerable scope for data-partitioning methods to be very effective in pruning the search space for efficient nearest-neighbor search (as is well-known from the practice).

2.3.3 Hyper-Sphere MBRs

The analysis above applies to index methods whose MBRs are hyper-cubes. There exists another group of index structures, however, such as the TV-tree, M-tree, VP-tree and SR-tree (see also the previous section), which use MBRs in the form of hyper-spheres. In an optimal structure, each block consists of the center point C and its $m-1$ nearest neighbors (where m again denotes the average number of data points per block). Therefore, the MBR can be described by the NN-sphere $nn^{sp,m-1}(C)$ whose radius is given by $nn^{dist,m-1}(C)$. If we now use a Minkowski sum to transform this region, we enlarge the MBR by the expected NN-distance $E(nn^{dist})$. The result is a new hyper-sphere given by

$$MinkSum\left(mbr_i, E(nn^{dist})\right) = sp^d\left(C, nn^{dist,m-1}(C) + E(nn^{dist})\right)$$

The probability, that block i must be visited during a NN-search can be formulated as:

$$P_{visit}(i) \;\geq\; Vol\left(sp^d\left(C, nn^{dist,m-1}(C) + E(nn^{dist})\right) \cap \Omega\right)$$

(this is a bound and not an equality since not all structures are optimal). Since $nn^{dist,i}$ does not decrease as i increases (that is, $\forall j > i:\; nn^{dist,j} \geq nn^{dist,i}$), another lower bound for this probability can be obtained by replacing $nn^{dist,m-1}$ by $nn^{dist,1} = nn^{dist}$:

$$P_{visit}(i) \;\geq\; Vol\left(sp^d\left(C, 2 \cdot E(nn^{dist})\right) \cap \Omega\right)$$

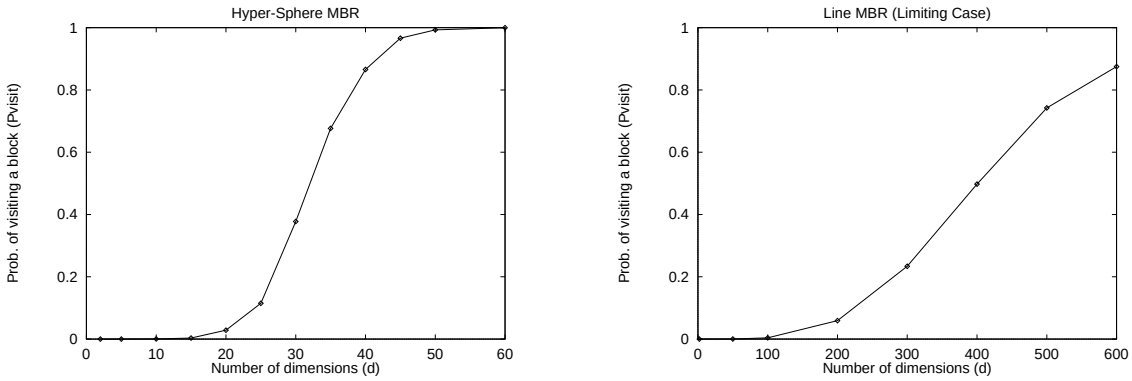
In order to obtain the probability of accessing a block during the search, we average the above probability over all center points $C \in \Omega$:

$$P_{visit}(i) \;\geq\; \iint\limits_{C\in\Omega} Vol\left(sp^d\left(C, 2 \cdot E(nn^{dist})\right) \cap \Omega\right) dC$$

Finally, the number of blocks which must be visited can be bounded as follows:

$$M^{visit} \;=\; \frac{N}{m} \cdot P_{visit}(i) \;\geq\; \frac{N}{m} \cdot \iint\limits_{C\in\Omega} Vol\left(sp^d\left(C, 2 \cdot E(nn^{dist})\right) \cap \Omega\right) dC$$

Figure 4 (a) shows that the percentage of blocks visited increases rapidly with the dimensionality, and reaches 100% with $d = 45$. This is a similar pattern to that observed above for hyper-cube MBRs. For $d = 26$, the critical performance threshold of 20% is already exceeded, and a sequential scan will perform better in practice, on average.



(a) (b)

Figure 4: The probability that a block must be visited if the MBR is: (a) a hyper-sphere; (b) a line.

2.3.4 Line MBRs (Limiting Case)

Given the results of the previous two sections, the difficulties of high dimensionality might be overcome either by increasing the number of points, or by reducing the extension of the MBR. Since real data sets are of given size, we focus here on the second possibility.

The case of only a single data point per block is not interesting since, in a tree structure, the problem is then simply shifted to the next level of the tree. Therefore, the smallest possible MBR is achieved when only two points are stored on each block. In this case, the optimal MBR is the line joining these two points. Given the best possible split strategy—in each block, the closest two points are stored—the average length of the line/MBR is bounded below by the average NN-distance. Following the same analytic techniques as before, we can estimate the number of blocks visited. Figure 4 (b) shows these results. In this limiting case, the 20% threshold is exceeded when $d \geq 280$.

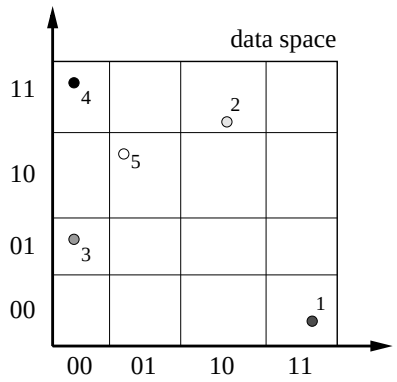
An important consequence of this limiting case is the following. *Even with an optimal index structure, data partitioning methods for nearest-neighbor searches will, as dimensionality increases, always degenerate to visiting the entire data set.* This result applies to the case of uniformly-distributed d-dimensional data, and also (we conjecture) to the case of arbitrarily-distributed data of fractal dimension d.

3 The Vector-Approximation File (VA-File)

As shown above, nearest-neighbor search tends towards linear complexity as dimensionality increases. Given this starting point, we propose here a search structure based not on data partitioning, but rather upon adaption and improvement of a fundamentally linear algorithm. The *vector approximation file*

b	number of bits per approximation	b_j	number of bits in dimension j
$m_j[k]$	k-th mark in dimension j	$r_{i,j}$	region into which $\vec{p}_i$ falls in dim j
l_i, u_i	bounds: $l_i \leq L_s(\vec{q}, \vec{p}) \leq u_i$	$L_p(\vec{a}, \vec{b})$	weighted distance function

Table 2: Notational summary for the VA-File method



		vector data	VA-File
1	●	0.9 0.1	11 00
2	○	0.6 0.8	10 11
3	●	0.1 0.4	00 01
4	●	0.1 0.9	00 11
5	○	0.3 0.7	01 10

Figure 5: Building the VA-File

(VA-File) is an array of compact, geometric approximations to data points. These smaller approximations are scanned, and each approximation determines a lower and an upper bound on the distance between its data point and the query. These bounds frequently suffice to filter most of the vectors from the search. IO costs are incurred only for those few candidates remaining after the filtering step.

This section describes the VA-File method and is organized as follows. Below, the VA-File itself is described. Section 3.2 describes the use of approximations to derive lower and upper bounds, and, based on these bounds, Section 3.3 presents two nearest-neighbor search algorithms. The notation used in this section is summarized in Table 2.

Assume N is the number of data points and d the number of dimensions. We use $i \in \{1, \ldots, N\}$ to range over data points, and $j \in \{1, \ldots, d\}$ over dimensions. Hence, $\vec{p}_i$ denotes an individual data point, and $p_{i,j}$ the j-th component within $\vec{p}_i$. Let a_i denote the approximation of $\vec{p}_i$, and let b_j be the number of bits per approximation in dimension j. The weighted metric $L_p(\vec{a}, \vec{b})$ is defined as:

$$L_p(\vec{a}, \vec{b}) = \left(\sum_{j=1}^{d} (w_j \cdot |a_j - b_j|)^p \right)^{1/p}$$

3.1 Structure

For each vector $\vec{p_i}$, an approximation is derived (see Figure 5). The first step is to assign a number of bits b_j to allocate to each dimension j. We use here an even distribution. Hence, the relationship between the total number of bits (b) and the number of dimensions (d) determines b_j as follows:

$$b_j = \left\lfloor \frac{b}{d} \right\rfloor + \begin{cases} 1 & \text{if } j \leq (b \bmod d) \\ 0 & \text{otherwise} \end{cases}$$

Typically b_j is a small integer: between 4 and 6.

The number of bits in each dimension is used to determine marks on the corresponding axes, as illustrated in Figure 5, left. In particular, there are 2^{b_j} partitions along dimension j, requiring $2^{b_j}+1$ marks. The first ($m_j[0]$) and the last mark ($m_j[2^{b_j}]$) correspond to the minimum and maximum value in this dimension. The remaining marks ($m_j[1], \ldots, m_j[2^{b_j}-1]$) are computed in order to partition the dimension into 2^{b_j} more or less equally-full partitions. The resulting set of marks is kept constant while insert, update and delete operations are performed, and is recomputed if the difference of the number of points in the partitions exceeds a given (tune-able) threshold. These techniques for establishing and maintaining marks are similar to those used for grid files [NHS84] or for partitioned-hashing schemes [Ull88]. Techniques for maintaining equi-depth approximate histograms can also be applied here [GMP97].

An approximation a_i for the data point p_i is generated as follows. Let the 2^{b_j} partitions in dimension j be numbered $0, \ldots, 2^{b_j}-1$. Then, let $r_{i,j}$ be the number of the partition into which $p_{i,j}$ falls. A point falls into a partition only if it lies between the lower and upper bounds of that partition:

$$m_i[r_{i,j}] \leq p_{i,j} < m_i[r_{i,j+1}]$$

The approximation a_i is simply the concatenation of the binary b_j-bit patterns for each partition in turn. The complete VA-File is an array of all these approximations, as illustrated in Figure 5, right. Notice that, unlike data-partitioning approaches such as grid file [NHS84], R-trees [Gut84] and their variants, data is not partitioned and clustered.

Given the set of marks, the data space is divided into 2^b cells. In practice, approximations frequently consists of more than 100 bits. As such, the number of cells exceeds the number of data points, and the vast majority of cells are empty.

3.2 Bounds

Based on approximations, we now derive simple bounds on the distance between a query point and a vector. Assume a query $\vec{q}$ and a distance function L_p, for some p. An approximation a_i determines a

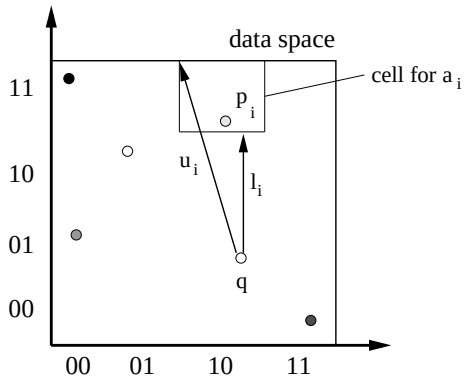


Figure 6: Lower and upper bounds for $L_p(\vec{q}, \vec{p_i})$ given a_i guaranteeing $l_i \leq L_p(\vec{q}, \vec{p_i}) \leq u_i$

lower bound l_i and an upper bound u_i such that:

$$l_i \leq L_p(\vec{q}, \vec{p_i}) \leq u_i$$

This is sketched in Figure 6. Intuitively, a_i contains sufficient information to determine the cell in which $\vec{p_i}$ lies. The lower bound l_i is simply the shortest distance from the query to a point in that cell. Similarly, the upper bound u_i is the longest distance to a point in that cell.

Formally, l_i and u_i are derived as follows. As with data points, a query $\vec{q}$ consists of components q_j and these fall into partitions numbered $r_{q,j}$. Also, given a_i, components $r_{i,j}$ are easily extracted. Based on this information, the bounds l_i and u_i are determined by the equations:

$$l_i = \left(\sum_{j=1}^{d} (w_j l_{i,j})^p \right)^{1/p} \qquad \text{where} \quad l_{i,j} = \begin{cases} q_j - m_j[r_{i,j}+1] & r_{i,j} < r_{q,j} \\ 0 & r_{i,j} = r_{q,j} \\ m_j[r_{i,j}] - q_j & r_{i,j} > r_{q,j} \end{cases}$$

$$u_i = \left(\sum_{j=1}^{d} (w_j u_{i,j})^p \right)^{1/p} \qquad \text{where} \quad u_{i,j} = \begin{cases} q_j - m_j[r_{i,j}] & r_{i,j} < r_{q,j} \\ \max\left(q_j - m_j[r_{i,j}], m_j[r_{i,j}+1] - q_j\right) & r_{i,j} = r_{q,j} \\ m_j[r_{i,j}+1] - q_j & r_{i,j} > r_{q,j} \end{cases}$$

3.3 Two Search Algorithms

The result of a nearest neighbor query must be the k data points closest to the query $\vec{q}$. This section describes two different search algorithms for finding those vectors. Both algorithms are based on the same underlying structure and bounds, as described above.

An important property of bounds presented above is the following. Given a query point, it is possible to precompute the distance, in each dimension, from the query point to each of the partition points.

```
VAR ans:  ARRAY OF INT; dst: ARRAY of REAL;
    a: ARRAY of Approx;  p⃗: ARRAY of Vector;

FUNC InitCandidate(): REAL;
VAR j: INT;
   FOR j := 1 TO k DO
      dst[j] := MAXREAL;
   RETURN MAXREAL;
END-FUNC InitCandidate;

FUNC Candidate(δ: REAL; i: INT): REAL;
   IF δ < dst[k] THEN
      dst[k] := δ; ans[k] := i;
      SortOnDst(ans, dst, k);
   RETURN dst[k];
END-FUNC Candidate;

PROC VA-SSA(q⃗: Vector);
VAR i: INT; l_i, δ: REAL;
   δ := InitCandidate();
   FOR i := 1 TO N DO
      l_i := GetBounds(a_i, v⃗_q);
      IF l_i < δ THEN
         δ := Candidate(L_p(p⃗_i, q⃗), i);
   END-FOR;
END-PROC VA-SSA;
```

Figure 7: The simple-search algorithm: *VA-SSA*

This eliminates the need to square values during the search itself, and reduces the CPU costs of the algorithms described below. This leads to the surprising fact that the VA-File search algorithms can consume less CPU resources than a scan algorithm. All costs associated with an individual query (including these pre-processing costs) are included in the results presented in the next section.

3.3.1 A 'Simple-Search' Algorithm (VA-SSA)

The simple-search algorithm, known as *VA-SSA*, is described in pseudo code in Figure 7. Arrays `ans` and `dst` are maintained of the nearest k vectors encountered so far, and their distances to $\vec{q}$, respectively. These are sorted in order of increasing distance. The approximations are scanned linearly. A vector is a candidate whenever less than k vectors have been encountered, or whenever the lower bound l_i is less than the distance of the k-th vector currently in the answer set. The actual distance based on L_p is evaluated only for these candidate vectors, hence only these candidates are visited. Thus, the VA-File is used as a simple filter.

The major advantages of this algorithm are simplicity, a low memory overhead, and that vectors are visited sequentially (i.e. there are no long, random seeks). On the other hand, the performance of

```
PROC VA-NOA(q⃗: Vector);
VAR i: INT; δ, l_i, u_i: REAL; Heap: HEAP; Init(Heap);

(* PHASE - ONE *)
   δ := InitCandidate();
   FOR i := 1 TO N DO
      l_i, u_i := GetBounds(a_i, q⃗);
      IF l_i <= δ THEN
         δ := Candidate(u_i, i);
         InsertHeap(Heap, l_i, i);
   END-FOR;

(* PHASE - TWO *)
   δ := InitCandidate();
   l_i, i := PopHeap(Heap);
   WHILE l_i < δ DO
      δ := Candidate(L_p(p⃗_i, q⃗), i);
      l_i, i := PopHeap(Heap);
   END-WHILE;
END-PROC VA-NOA;
```

Figure 8: The near-optimal search algorithm: *VA-NOA*

simple-search depends upon the ordering of the vectors and approximations. Despite this potential difficulty, Section 4.2 shows that this algorithm performs well in practice.

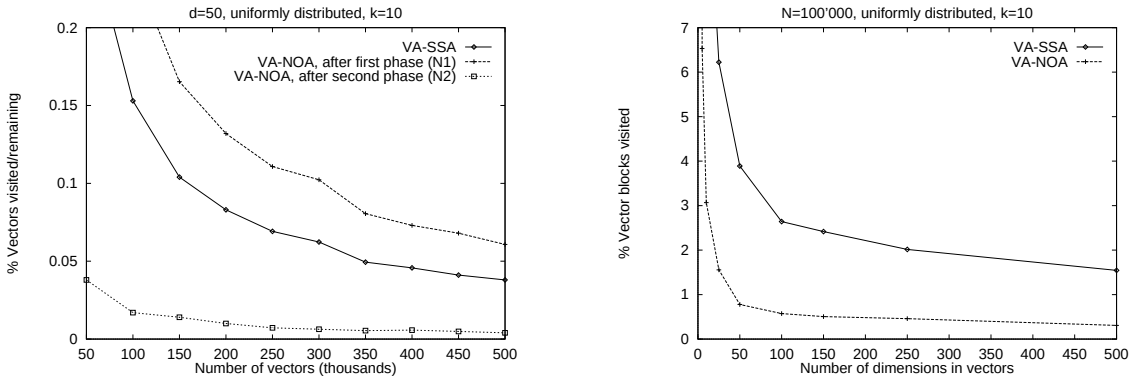
3.3.2 A Near-Optimal Search Algorithm

Next, we present an algorithm (*VA-NOA*) that is near-optimal in the sense that it minimizes the number of vectors visited, i.e. it minimizes IO operations. In fact, this algorithm is optimal for all but a few highly-degenerate cases (so far, there is only one situation known, for which *VA-NOA* is not optimal[1]). The algorithm operates in two phases, and is sketch in Figure 8.

During the first phase, the approximations are scanned, and the bounds l_i and u_i are computed for each vector. Then the following filtering step is applied. Assume δ is the k-th largest upper bound encountered so far. If an approximation is encountered such that l_i exceeds δ, then $\vec{p_i}$ can be eliminated since k better candidates have already been found. In our practical experiments, between 95% and 99% of the vectors were eliminated during this first filtering step. Let N_1 denote the number of candidates remaining after this first phase. N_1 proved to be sub-linear in the database size. The space overhead of this algorithm is proportional to N_1.

During the second phase, the answer set is determined by visiting the vectors themselves in increasing order of l_i. Not all the remaining candidates must be visited. Rather, this phase ends when a lower bound is encountered which exceeds or equals the k-th distance in the answer set. The data structure

[1]The degenerate, non-optimal conditions are the following. Assume $\vec{p_i}$ is the final member of the answer set found by *VA-NOA*. Further, let δ be $L_p(\vec{q}, \vec{p_i})$. Then non-optimality occurs only if: 1) $l_i = \delta$, 2) the lower bound of the previously-visited vector is also δ, 3) no vector is visited by *VA-NOA* after $\vec{p_i}$, and 4) no vector in the answer set has a distance to the query larger than δ. These conditions are exceedingly improbable in practice.



(a) (b)

Figure 9: (a) Vector selectivity for the VA-File as a function of the database size; and (b) block selectivity as a function of dimensionality.

supporting this procedure is a heap-based priority queue [Sed83]. Let N_2 denote the number of steps performed during this phase. Then the total number of additional steps for this second phase is roughly $N_2 \log_2 N_1$. In particular, only the N_2 elements necessary to complete the search are sorted.

4 Performance Evaluation

To show the practical relevance of the VA-File, we performed many evaluations based on synthetic data sets and real data sets. The synthetic data set was obtained by randomly generating the desired number of points in the data space $\Omega = [0, 1]^d$. The real data was a set of 45-dimensional feature vectors extracted form an image database consisting of over 50'000 images. The similarity method for color distribution described by Stricker and Orengo [SO95] generates 9-dimensional feature vectors for each image. A newer approach treats five overlapping parts of images separately, and generates a 45-dimensional feature vector for each image. In all experiments reported here, the L_2 metric with all weights equal to 1 was chosen. The number of nearest neighbors to return was always ten, i.e. $k = 10$. The experimental results presented in this section are computed on a Sun SPARCstation 4 with 64 MBytes of main memory. We used the original implementation of the X-tree [BKK96], and an optimized version of the original R*-tree [BKSS90]. Due to a limitation of the code, the X-tree could only maintain data with less than 32 dimensions. The following four search structures were evaluated. The VA-File, the R*-tree, the X-tree and a simple sequential scan. The sequential scan retrieved data in blocks of 400K, and the data for all the methods reported was stored on a local disk.

4.1 Selectivity Experiments for the VA-File

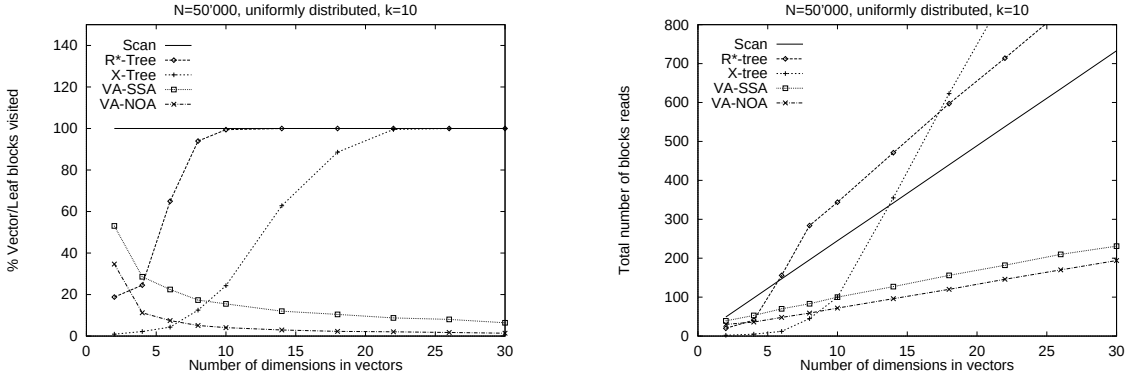
Two sets of selectivity experiments for the VA-File approach were performed. Selectivity was measured firstly as a function of the database size, and secondly as a function of the number of dimensions. The database consisted of (synthetic) uniformly-distributed data points. Figure 9 (a) shows improved vector selectivity as the number of data points increases ($d = 50$). At $N = 500'000$, less than 0.1% of the vectors remain after the first phase of VA-NOA, and only about 0.02% of the vectors are actually visited by this algorithm. At $N = 50'000$, only 19 vectors are visited, while at $N = 500'000$ only 20 vectors are visited. Thus, almost the minimum number of necessary visits (i.e. $N_2 = k = 10$) was achieved. The selectivity of the simple algorithm VA-SSA is about five times smaller but still very high. At $N = 50'000$, 134 vectors are visited, while at $N = 500'000$, 190 vectors are visited.

Figure 9 (b) shows the percentage of blocks visited as a function of dimensionality. This graph directly reflects the estimated IO cost of the VA-File. For low-dimensional data spaces ($d < 5$), our threshold of 20% is not reached with both algorithms, and other methods (e.g., R*-Tree, X-Tree, or scan) may generally perform better. However, for high-dimensional spaces, the VA-File offers significant advantages over either data-partitioning or scan-based methods. For $d \geq 35$, only around 1% (VA-NOA) and 5% (VA-SSA) data blocks are accessed. Moreover, performance remains equally good and even improves as dimensionality increases.

4.2 Performance Comparison with Other Methods

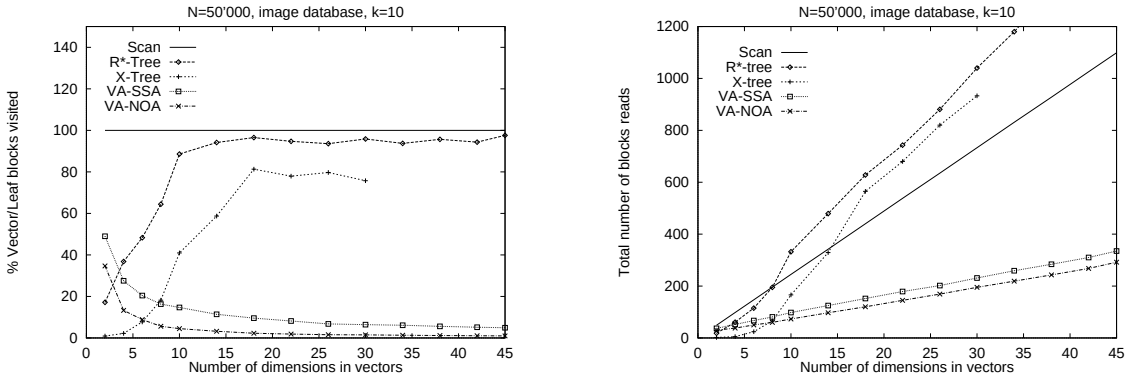
The performance of NN-search in tree based methods is usually estimated by counting the number (or the fraction) of blocks visited. The CPU cost is considered to be marginal when compared to the IO costs of secondary-memory accesses. Figures 10 and 11 compare the performance of the different access methods as a function of dimensionality for a uniformly-distributed data set (Figure 10), and for a real data set (Figure 11). The left-hand sides show the percentage of blocks visited, and the right-hand sides show the total number of blocks visited, both as a function of dimensionality. The size of a block was always 8192 bytes. As our analysis predicts, the performance of the data-partitioning methods degrades severely as dimensionality increases. In practice based on our 20% threshold, these data-partitioning methods should not be considered for data sets of dimensionality greater than around ten (see Figure 10 (a) and Figure 11 (a)). Ultimately, these methods access even more blocks than a sequential scan. For $d > 10$ and the metrics shown, the VA-File outperforms all other methods. These conclusions hold in general for both data sets.

Notice that there is an apparent anomaly in Figure 11 (a). In particular, this graph appears to contradict that of Figure 3 (b) (which predicts that the graph must reach 100%). The explanation for this is the following. There is a hidden assumption in our analysis which comes into play here. In



(a) (b)

Figure 10: (a) Block selectivity and, (b) the total number of blocks accessed as a function of the number of dimensions. Synthetic data set (uniformly distributed).



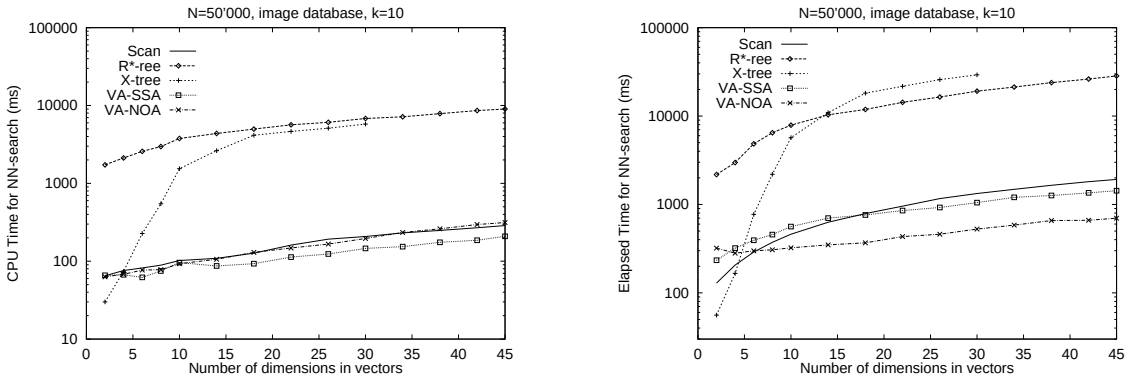
(a) (b)

Figure 11: (a) Block selectivity and, (b) the total number of blocks accessed as a function of the number of dimensions. Real data set.

particular, the analysis assumes that all the data exists within a closed data space. In practice, real data sets have outliers. The blocks on which these outliers reside are visited only very infrequently, implying that 100% of the blocks are not, in fact, visited in practice.

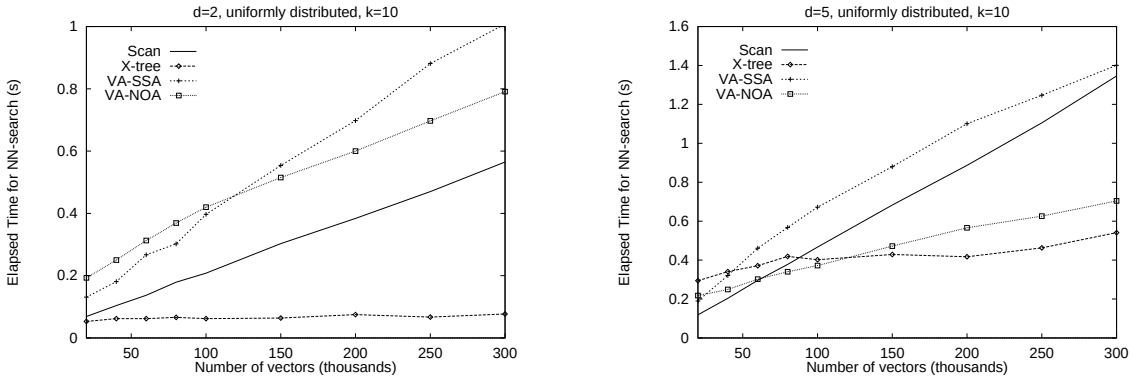
4.3 CPU and Wall-Clock Experiments

To demonstrate the practical relevance of the VA-File, we performed a number of timing experiments based on the real image data set. In these experiments, the number of indexing dimensions was varied. Figure 12 (a) shows the CPU time, and Figure 12 (b) shows elapsed time (also called wall-clock time) of a nearest neighbor search for each of the four methods considered. Notice that the scale of the



(a) (b)

Figure 12: CPU time (a) and wall-clock time (b) for the image database



(a) (b)

Figure 13: Wall-clock time as function of the database size, for (a) 2 dimensions, and (b) 5 dimensions

y-axis is logarithmic in this figure. In comparison with the tree based structures, the CPU cost of the VA-File is up to one order of magnitude smaller, in fact, the CPU time never exceeded 0.5 seconds. Surprisingly, despite its heavy use of bit operations, the CPU costs of the VA-File methods can even be less than those of the scan. This is because of the additional arithmetic which must be performed by the scan. Consequently, even if the whole database fits into main memory, the VA-File can offer a performance improvement over a well-tuned scan.

For very large databases, the feature vectors of the objects do not fit into main memory and IO operations become the limiting factor. The wall-clock experiments presented in Figure 12 (b) are performed on the local, uncached disk of our test machine. In low-dimensional data spaces, the sequential scan ($5 \leq d \leq 10$) and the X-tree ($d < 5$) produce least disk operation and execute the nearest neighbor search fastest. In high-dimensional data spaces, that is $d > 10$, the VA-File outperforms all other methods.

Figure 12 (b) also shows that the tree-based methods are unsuitable if the data consists of more than around 5 dimensions. In particular, even in very low-dimensional spaces the R*-tree is 10 times slower than the sequential scan. Although fastest at low dimensionality, the performance of the X-tree degrades rapidly, and ultimately becomes even worse than that of the R*-tree.

Finally, Figure 13 illustrates relative wall-clock times at low dimensionality in greater detail. These wall-clock experiments were performed on a uniformly distributed data set with 2 and 5 dimensions, respectively, and the database size was varied up to 300'000 points [notice that the previous experiments were performed on a fixed data-set size of 50'000 points]. For the 2-dimensional data set (Figure 13 (a)), the X-tree performed best and determined the nearest neighbors up to 10 times faster than the scan and the VA-File. As illustrated in Figure 13 (b), however, the advantage of the X-tree can be lost in the case of 5-dimensional data points. With the increasing number of points, the performance of the X-tree method improves since the data space becomes increasingly dense, and thus the influence of the larger dimensionality is diminished. However, there is little difference between the methods, and, for moderate database sizes, the VA-File may even to be preferred for this low-dimensional case.

5 Related Work

There is a considerable amount of related work in the theory and practice of indexing high-dimensional vector spaces. The general model of using abstract features as the basis for indexing in, for example, multimedia applications has been described by Faloutsos [Fal96]. We will not focus here on the array of multi-dimensional indexing techniques which have been proposed in the literature, except to say that many of these techniques have been referenced within the body of this text, as appropriate. An interesting generalization of entire classes of data-partitioning access methods has been proposed by Hellerstein, Naughton and Pfeffer [HNP95].

The 'indexability' results of Hellerstein, Koutsoupias and Papadimitriou [HKP97] are directly related to the analysis of Section 2. For range queries, these authors presented a minimum bound on the 'access overhead' of $B^{1-\frac{1}{d}}$, which tends toward B as dimensionality increases (B denotes the size of a block). This result is consistent with the average-case results established here for nearest-neighbor queries with either hyper-cubic or hyper-spherical MBRs.

There is also extensive previous work on cost-model–based analysis, similar to the results established here. Early work in this area includes [FBF77, Cle79, Spr91], but failed to address the specific difficulties of high dimensionality. Berchtold, Böhm Keim and Kriegel have more recently addressed the issue of high dimensionality, and, in particular, the boundary effects which impact the performance of high-dimensional access methods. Our use of Minkowski Sums, much of our notation, and the

MBR	20th percentile	90th percentile	Case
Hyper-Cubes	18	28	$N = 1'000'000$
Hyper-Spheres	26	43	
Lines (limiting case)	280	600+	

Table 3: Summary of analytical results based on Figures 3 (b), 4 (a) and 4 (b)

structure of our analysis for hyper-cubic MBRs follow their analysis. However, our analysis extends theirs by considering not just hyper-cubic MBRs, but also hyper-spherical MBRs, and, as the limiting case, linear MBRs.

The VA-File method which we proposed has aspects in common with a number of existing methods. The way the data space is partitioned is similar to techniques used for grid files [NHS84] or for partitioned-hashing schemes [Ull88]. However, our use of this partitioning is more akin to its use within signature files [Fal85, FC87], although signature files are generally useful for partial match queries. Here, these techniques have been adapted to the requirements of nearest-neighbor queries. To the best of our knowledge, the structure and performance of the VA-File (or a similar structure) have not been presented previously.

6 Conclusion and Future Work

We provided an analysis of the performance of data-partitioning index methods (R-Tree, X-Tree, M-Tree, etc.) at low, moderate, and high dimensionality. We have focussed on nearest-neighbor queries since these are frequently critical in the applications where similarity search in high-dimensional vector spaces arise naturally (such as multimedia systems, decision support, data mining and geographical systems). We have focussed upon three classes of methods: those whose MBRs are hyper-cubes, those whose MBRs are hyper-spheres, and those whose MBRs are lines. The first two of these cases are of practical importance since there exist many methods which assume such MBRs. The case of lines, however, is of theoretical importance since it places a bound on the possible average performance of any data-partitioning method. Finally, while our bounds for hyper-cubes are tighter than our bounds for hyper-spheres, we do not feel that that necessarily indicates that hyper-spheres are a better choice for data-partitioning methods. Rather, this result stems from the fact that we were able to make use of stronger information in obtaining bounds for the hyper-cubes case, than we were for the hyper-spheres case.

Given this analytical basis, we then described the VA-File, an approximation-based organization for high-dimensional data sets, and have provided a performance evaluation for this method. Although

deceptively simple, experimental results based on both selectivity and wall-clock experiments are very promising. At moderate and high dimensionality, and for moderate and large data sets, the VA-File method can outperform any other method known to the authors. We have also shown that performance improves as dimensionality increases (whereas the performance of data-partitioing methods degrades as dimensionality increases). The experiments reported here offer one of the most comprehensive studies of the practical impact of high dimensionality on a variety of similarity search methods.

Although presented wholly in terms of the L_p norms, the results presented here depend relatively weakly upon which particular distance metric is chosen. Rather, the requirement is that a lower and a upper bound can be derived based on the geometrical approximation which is used.

The simple, flat structure of the VA-File offers a number of important advantages in addition to the performance issues discussed in detail here. It is possible to integrate search in VA-File structures over multiple vector spaces, and also over signature files for non-metric spaces. This capability can be used for conjunctive queries. It is also possible to include weights in the distance metric, thereby accommodating weighted search and, as a special case, partial match. Weighted search is necessary to support relevance feedback mechanisms. Also, extensions to the method incorporating parallelism, distribution, concurrency and recovery (all of which are non-trivial for tree-based methods) are simplified by the simple flat structure. These issues are all subject to future work.

Acknowledgments. This work has been partially funded in the framework of the European ESPRIT project HERMES (project no. 9141) by the Swiss *Bundesamt für Bildung und Wissenshaft* (BBW, grant no. 93.0135), and partially by the ETH cooperative project on *Integrated Image Analysis and Retrieval*. We also thank the authors of [BKSS90, BKK96] for making their R*-tree and X-tree implementations available to us.

References

[BBB+97] S. Berchtold, C. Böhm, B. Braunmüller, D.A. Keim, and H.-P. Kriegel. Fast parallel similarity search in multimedia databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1–12, Tucson, USA, 1997.

[BBKK97] S. Berchtold, C. Böhm, D.A. Keim, and H.-P. Kriegel. A cost model for nearest neighbour search. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, pages 78–86, Tucson, USA, 1997.

[BF79] J.L. Bentley and J.H. Friedman. Data structures for range searching. *ACM Computing Surveys*, 11(4):397–409, Decmeber 1979.

[BKK96] S. Berchtold, D.A. Keim, and H.-P. Kriegel. The X-tree: An index structure for high-dimensional data. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 28–39, 1996.

[BKSS90] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 322–331, Atlantic City, NJ, 23–25 May 1990.

[Cle79] J.G. Cleary. Analysis of an algorithm for finding nearest-neighbors in euclidean space. *ACM Transactions on Mathematical Software*, 5(2), 1979.

[CPZ97] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, Athens, Greece, 1997.

[Csi95] A. Csillaghy. Retrieving information from digital solar radio spectrograms. In *Coronal Magnetic Energy Releases (Lecture Notes in Physics)*. Springer Verlag, 1995.

[Csi97] A. Csillaghy. *Information extraction by local density analysis: A contribution to content-based management of scientific data.* Ph.D. thesis, Institut für Informationssysteme, 1997.

[Dim97] A. Dimai. Differences of global features for region indexing. Technical Report 177, ETH Zürich, Feb. 1997.

[Fal85] C. Faloutsos. Access methods for text. *ACM Computing Surveys*, 17(1):49–74, March 1985. Also published in/as: "Multiattribute Hashing Using Gray Codes", ACM SIGMOD, 1986.

[Fal96] C. Faloutsos. *Searching Multimedia Databases By Content.* Kluwer Academic Press, 1996.

[FBF77] J.H. Friedman, J.L. Bentley, and R.A. Finkel. An algorithm for finding best-matches in logarithmic time. *TOMS*, 3(3), 1977.

[FC87] C. Faloutsos and S. Christodoulakis. Description and performance analysis of signature file methods for office filing. *ACM Transactions on Office Information Systems*, 5(3):237–257, July 1987.

[FK94] C. Faloutsos and I. Kamel. Beyond uniformity and independence: Analysis of R-trees using the concept of fractal dimension. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, 1994.

[FSN+95] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker. Query by image and video content: The QBIC system. *Computer*, 28(9):23–32, September 1995.

[GMP97] P. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 1997.

[Gut84] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 47–57, Boston, MA, June 1984.

[HKP97] J.M. Hellerstein, E. Koutsoupias, and C.H. Papadimitriou. On the analysis of indexing schemes. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, 1997.

[HNP95] J.M. Hellerstein, J.F. Naughton, and A. Pfeffer. Generalized search trees for database systems. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 1995.

[HS95] G.R. Hjaltason and H. Samet. Ranking in spatial databases. In *Proceedings of the Fourth International Symposium on Advances in Spatial Database Systems (SSD95)*, number 951 in Lecture Notes in Computer Science, pages 83–95, Portland, Maine, August 1995. Springer Verlag.

[KS97] N. Katayama and S. Satoh. The SR-tree: An index structure for high-dimensional nearest neighbor queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 369–380, Tucson, Arizon USA, 1997.

[LJF94] K.-I. Lin, H.V. Jagadish, and C. Faloutsos. The TV-tree: An index structure for high-dimensional data. *The VLDB Journal: The International Journal on Very Large Data Bases*, 3(4):517–549, October 1994.

[NHS84] J. Nievergelt, H. Hinterberger, and K.C. Sevcik. The grid file: An adaptable symmetric multikey file structure. *ACM Transactions on Database Systems*, 9(1):38–71, March 1984.

[Sam89] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, 1989.

[Sed83] R. Sedgewick. *Algorithms.* Addison-Wesley Publishing Company, 1983.

[SO95] M. Stricker and M. Orengo. Similarity of color images. In *Storage and Retrieval for Image and Video Databases, SPIE*, San Jose, CA, 1995.

[Spr91] R.F. Sproull. Refinements to nearest-neighbor search in k-dimensional trees. *Algorithmica*, 1991.

[Ull88] J.D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume 1. Computer Science Press, 1988.